Vérification de propriétés temporelles à la volée

Laurent Kaiser

Vérification de propriétés temporelles à la volée*

Laurent KAISER
CRIN, Bâtiment LORIA, Campus Scientifique BP 239
54506 Vandoeuvre les Nancy, France.
E-mail : lkaiser@loria.fr

Résumé
De plus en plus de systèmes intègrent des contraintes temporelles dans leur spécification. Il est important de pouvoir vérifier qu'une ou plusieurs propriétés désirées sont bien respectées. Néanmoins, cette vérification est confrontée au problème de l'explosion combinatoire engendrée par les approches exhaustives. C'est dans le but d'éviter cette explosion et donc de pouvoir traiter des problèmes plus complexes que nous présentons dans cet article une méthode de vérification à la volée de propriétés temporelles.

Mots-clés : Vérification à la volée, contrainte temporelle, automate à entrées-sorties

1. Introduction

La vérification de propriétés temporelles est une problématique très présente dans les domaines de l'interopérabilité, de la communication, des systèmes embarqués, etc.Il est important, si l'on désire utiliser des contraintes temporelles dans une spécification de pouvoir vérifier qu'une ou plusieurs propriétés désirées sont bien vérifiées. Une préoccupation bien connue est la détermination de l'existence de comportements satisfaisant une propriété donnée. On distingue des approches basées sur la simulation, ainsi que des approches basées sur le calcul exhaustif d'un graphe de comportement. Mais, la vérification est confrontée au problème d'explosion combinatoire du graphe exploré. Les techniques de vérification à la volée permettent d'analyser une propriété sans avoir à construire le graphe en totalité [4]. Sokolsky [6] indiquait que ces techniques n'avaient jamais été appliquées à des systèmes temps réel auparavant et proposait une méthode basée sur des extensions temporelles du mu-calcul. Nos travaux s'appuient sur des modèles d'automates temporisés, sur une méthode de vérification exhaustive que nous avons développée [5] ainsi que sur les résultats de J. Toussaint [7], [8]. Nous proposons dans cet article une nouvelle technique de vérification de propriétés temporelles. Cette méthode permet de trouver à la volée l'existence de chemins satisfaisants une propriété donnée en diminuant dans certains cas l'explosion combinatoire.

Dans cet article, nous ferons tout d'abord un bref rappel sur le modèle que nous avons utilisé : les *TIOSM*. Ce modèle fait partie de la famille des automates temporisés d'Alur et Dill [1], et permet de décrire les systèmes à entrées-sorties. Nous présenterons ensuite l'algorithme de vérification à la volée de propriétés temporelles que nous avons développé.

2. Modélisation

Dans cette section, nous allons tout d'abord présenter le modèle des *TIOSM*, puis nous verrons comment modéliser une propriété, et enfin, nous définirons les règles utilisées pour rechercher la propriété.

2.1. Rappel sur les TIOSM (Timed Input Output State Machine)

Définition 2.1 *Un TIOSM* [5] *est un tuple* $M = (S(M), L(M), C(M), s_0(M), C_0(M), T(M))$ *où* $S(M)$ *est un ensemble fini et non vide d'états,* $L(M)$ *est un ensemble fini d'interactions,* $C(M)$ *est un ensemble fini d'horloges,* $s_0(M)$ *est l'état initial de* M*,* $C_0(M) \subseteq C(M)$ *est un ensemble fini d'horloges remises à zéro lors de l'initialisation et* $T(M)$ *est un ensemble fini de transitions.*

Une transition $t \in T(M)$ *est définie par* $t = (s(t), m, D(t), Z(t), d(t))$ *où* $s(t)$ *est l'état source,*$m \in (\{!, ?\} \times L(M)) \cup \{\tau\}$ *est une action interne* (τ)*, une émission* ($!a$) *ou une reception* ($?b$)*,* $D(t)$ *est un*

* *Nous remercions Ousmane KONE pour la direction de ces travaux.*

ensemble fini de contraintes temporelles du type $c \in [a,b]$ (avec $c \in C(M), (a,b) \in (\mathbb{Q} \times \mathbb{Q})$), $Z(t)$ est un ensemble fini d'horloges à remettre à zéro après le tir de t, et, $d(t)$ est l'état destination (cf. figure 1).

2.2. Modélisation avec les TIOSM

Pour pouvoir vérifier une propriété P dans une spécification S, nous allons tout d'abord représenter S sous la forme d'un TIOSM T_s, puis faire de même pour P en la représentant par le TIOSM T_p.

Dans la suite de cet article nous considérerons qu'une propriété est vérifiée si on peut atteindre un état d'acceptation de l'automate représentant cette propriété.

Définition 2.2 (Etat d'acceptation)
On définit $Accept(T_p)$ ($Accept(T_p) \subseteq S(T_p)$), l'ensemble fini des états d'acceptation du TIOSM T_p correspondant aux états pour lesquels la propriété P est satisfaite (dans la figure 1, on définit l'ensemble des états d'acceptation de TP par : $Accept(TP) = \{S10\}$).

2.3. Règles de construction

Pour permettre la recherche d'un chemin dans T_s vérifiant la propriété T_p, nous allons définir des règles permettant de construire le *TIOSM* $T_{s||p} = T_s||T_p$ (avec $S(T_{s||p}) \subseteq S(T_s) \times S(T_p)$, $L(T_{s||p}) = L(T_s) \cup L(T_p)$, $C(T_{s||p}) = C(T_s) \cup C(T_p)$ et $C_0(T_{s||p}) = C_0(T_s) \cup C_0(T_p)$). Cette opération de construction est proche de la notion de composition de CCS, et, les règles ci-dessous ont été écrites de manière à pouvoir *trouver* la propriété P quel que soit l'endroit où elle se trouve dans la spécification S :

- Règle R_0 : Cette règle sert à constuire le premier état de $T_{s||p}$ à partir des états initiaux de T_s et T_p (on considère que les deux TIOSM commencent en même temps).

$$s_0(T_{s||p}) = (s_0(T_s), s_0(T_p)) \in S(T_{s||p})$$

- Règle R_1 : Cette règle traite les cas où l'on ne peut tirer des transitions que dans T_s. En fait, il n'est pas possible d'*avancer* dans T_p.

$$\frac{\begin{array}{c}(s_1, s_2) \in S(T_{s||p}) \wedge \\ (s_1, \mu, D_1, Z_1, s_1') \in T(T_s) \wedge (s_2, \mu, D_2, Z_2, s_2') \notin T(T_p)\end{array}}{(s_1', s_2) \in S(T_{s||p}) \wedge ((s_1, s_2), \mu, D_1, Z_1, (s_1', s_2)) \in T(T_{s||p})}$$

- Règle R_2 : Cette règle sert lorsque l'on peut tirer simultanément une transition de T_s et T_p. On se trouve en fait dans la situation où une *partie* de la propriété P est vérifiée.

$$\frac{\begin{array}{c}(s_1, s_2) \in S(T_{s||p}) \wedge \\ (s_1, \mu, D_1, Z_1, s_1') \in T(T_s) \wedge (s_2, \mu, D_2, Z_2, s_2') \in T(T_p)\end{array}}{\begin{array}{c}(s_1', s_2') \in S(T_{s||p}) \wedge ((s_1, s_2), \mu, D_1 \cup D_2, Z_1 \cup Z_2, (s_1', s_2')) \in T(T_{s||p}) \wedge \\ (s_1', s_2) \in S(T_{s||p}) \wedge ((s_1, s_2), \mu, D_1, Z_1, (s_1', s_2)) \in T(T_{s||p})\end{array}}$$

3. Méthode de vérification à la volée

La méthode de vérification temporelle que nous avons développée s'inspire de l'approche énumérative des RdPT. Nous y avons ajouté les techniques de vérification à la volée que nous avons étendues pour pouvoir prendre en compte des contraintes temporelles . Nous allons tout d'abord faire un rappel sur la méthode énumérative puis, nous présenterons notre algorithme.

3.1. Rappel sur la méthode énumérative

L'approche énumérative [2], [3], [7] consiste à construire l'ensemble du graphe d'accessibilité à partir d'une classe d'états initiale $C_0 = (s_0(T_s), s_0(T_p), Q_0)$.

Définition 3.1 (Classe d'états) *Une classe d'états $C = (s_1, s_2, Q)$ est définie par :*
- s_1 est l'état courant de l'automate T_s, et, s_2 est l'état courant de l'automate T_p
- Q est un système d'inéquations permettant de conserver les relations temporelles entre les diverses transitions sensibilisées dans l'état global $(s_1, s_2) \in S(T_{s||p})$. Ce système est composé de deux types d'inéquations ($\theta(t)$ est l'instant de tir de la transition t):

$$\alpha \leq \theta(t) \leq \beta \text{ : temps restant pour tirer } t$$

$\theta(t) - \theta(t') \leq \Delta$: *plus grande différence entre les instants de tir de t et de t'*
(Le système Q_0 contient les données temporelles sur les instants de tir des transitions lorsque l'on se trouve dans l'état global $(s_0(T_s), s_0(T_p))$)

Le calcul de la totalité du graphe des classes d'états sera terminé lorsqu'il ne restera plus de transitions tirables à traiter, ou, lorsque l'on ne retrouvera que des classes d'états déjà atteintes.

3.2. Algorithme de vérification à la volée

L'algorithme de notre méthode de vérification de propriétés temporelles à la volée utilise un mécanisme de parcours en profondeur avec retour arrière en cas d'echec.

Pour cela, on calcule tout d'abord la première classe d'état C_0. Ensuite à partir d'une classe d'états C, on construit et on parcourt en profondeur le graphe des classes d'états :

- Si on atteint une classe d'états $C' = (s_s, s_p, Q)$ où $s_p \in Accept(T_p)$ alors on aura trouvé un chemin vérifiant la propriété P (Le chemin est : $C_0 \rightarrow ... \rightarrow C_i \rightarrow ... \rightarrow C \rightarrow C'$).
- Si on atteint une classe d'états déja calculée, on en recherche une autre
- Si on ne peut atteindre une nouvelle classe d'états, on retourne en arrière (classe d'états C)
- Si toutes les classes d'états ont été calculées, alors la propriété P n'est pas vérifiée dans S

Notation 3.1
- *T_s et T_p sont les TIOSM modélisant respectivement la spécification et la propriété à vérifier*
- *$RelationTemporelle(\{(s^s, s^p), q^c\}, \mu, (s_2^s, s_2^p))$ calcule le nouveau système d'inéquations à partir de l'ancien système q^c et des contraintes temporelles associées au passage de (s^s, s^p) à (s_2^s, s_2^p)*
- *$ExisteSolution((s^s, s^p), q^c, (s_2^s, s_2^p))$ vérifie que les contraintes temporelles pour passer de (s^s, s^p) à (s_2^s, s_2^p) peuvent être vérifiées dans q^c*
- *$Successeur((s^s, s_p))$ cherche l'ensemble des états globaux atteignables à partir de l'état global (s^s, s^p) en respectant les règles R_0, R_1 et R_2*
- *P est une pile contenant l'ensemble des classes d'états à traiter*
- *$Calcule$ est l'ensemble des classes d'états déja calculées*
- *$Chemin$ est un chemin permettant de vérifiée la propriété P*

$P := \emptyset$; $Calcule := \emptyset$; $Chemin := \emptyset$
$q_0^c := RelationTemporelle(\emptyset, \emptyset, (s_0(T_s), s_0(T_p)))$
$Ajout(\{(s_0(T_s), s_0(T_p)), q_0^c\}, Calcule)$
$Empile(\{(s_0(T_s), s_0(T_p)), \emptyset\}, Chemin)$
$Empile(\{(s_0(T_s), s_0(T_p)), q_0^c, Successeur((s_0(T_s), s_0(T_p)))\}, P)$
`* Itération tant qu'il reste des classes d'états à traiter *`
`Tantque` $(P \neq \emptyset)$ `faire`
  $\{(s^s, s^p), q^c, l\} := Tete(P)$
  `Si` $(l \neq \emptyset)$ `alors`
    $\{(s_2^s, s_2^p), \mu\} := Tete(l)$
    $Depile(l)$
    $q_2^c := RelationTemporelle(\{(s^s, s^p), q^c\}, \mu, (s_2^s, s_2^p))$
    `Si` $(\{(s_2^s, s_2^p), q_2^c\} \notin Calcule \wedge ExisteSolution((s^s, s^p), q^c, (s_2^s, s_2^p))$ `alors`
`* On obtient une nouvelle classe d'états` $\{(s_2^s, s_2^p), q_2^c\}$ `*`
      $Empile(\{(s_2^s, s_2^p), q_2^c, Successeur((s_2^s, s_2^p))\}, P)$
      $Empile(\{(s_2^s, s_2^p), \mu\}, Chemin)$
`* Si le` s_2^p `est un état d'acceptation, alors on a vérifié la propriété *`
      `Si` $(s_2^p \in Accept(T_p))$ `alors` $Retourne(Chemin)$
  `FinSi`
  `Sinon`
    $Depile(P)$
    $Depile(Chemin)$
  `FinSi`
`Fin Tantque`
`* La propriété n'a pu être vérifiée *`
$Retourne(\emptyset)$

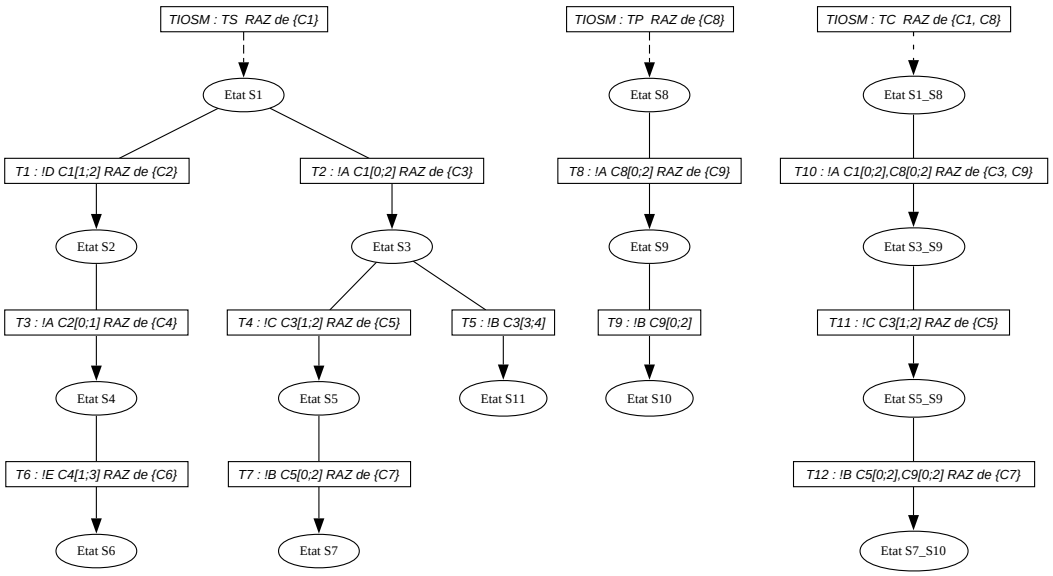

FIG. 1: *TS : représentant la spécification S, TP : représentant la propriété P et* $TC = TS||TP$ *: représentant un chemin vérifiant P dans S*

4. Conclusion

L'approche exhaustive utilisée pour vérifier des propriétés temporelles nécessite énormement de ressources car il faut calculer l'ensemble du graphe d'accessibilité. C'est pourquoi, nous avons proposé une méthode de vérification à la volée de propriétés temporelles qui tente d'éviter l'explosion combinatoire engendrée par l'utilisation d'une méthode exhaustive puisqu'elle ne consiste pas à calculer et mémoriser l'ensemble du graphe d'accessibilité. L'algorithme que nous avons présenté a été implanté dans l'outil *Xtiosm* [9], qui permet aussi de réaliser d'autres opérations sur les TIOSM (composition et validation de TIOSM, ...). Nous nous intéressons à modéliser et valider des protocoles à contraintes temporelles complexes.

Bibliographie

1. R. Alur, D.L. Dill. The Theory of Timed Automata. Theoretical Computer Science, 126(2), 1994.
2. B.Berthomieu, M.Diaz. Modeling and Verification of Time Dependent Systems Using Time Petri Nets. IEEE Transactions on Software Engineering. 3(17), 03/1991, pp 259-273.
3. L. Gallon. Le modèle réseaux de Petri temporisés stochastiques : extensions et applications Thèse de doctorat, Toulouse, 1997.
4. C. Jard, T. Jéron, J.C. Fernandez, L. Mounier. On the fly verification of finite transition systems. Formal Methods in System Design, 1:251-273, 1992
5. L. Kaiser, O. Koné. Une méthode de vérification d'interopérabilité temporelle. Colloque francophone RENPAR'9, Lausanne, 1997
6. O.V. Sokolsky, S.A. Smolka. Local Model Checking for Real-Time Systems. Computer Aided Verification. pp 211-224, Liège, 1995.
7. J. Toussaint. Modélisation temps réel réparties pour la validation de propriétés temporelles, Méthodologie de construction de modèles et algorithmes de validation. Thèse de doctorat, INPL, Nancy, 1997.
8. J. Toussaint, F. Simonot-Lion. Vérification formelle de propriétés temporelles d'une application distribuée temps réel. Real-Time Systems, RTS'97, pages 53-66, Paris (France), janvier 1997.
9. L. Kaiser. Interopérabilité temporelle à l'aide d'automates temporisés à entrée/sortie. Mémoire de DEA, Université Henri Poincaré, Nancy, 1996.